# Promoting Privacy

Sean Barnum, Cigital, Inc. [vita[3]]
Michael Gegick, Cigital, Inc. [vita[4]]

2005-09-14

Protecting software systems from attackers that may obtain private information is an important part of software security. If an attacker breaks into a software system and steals private information about a vendor's customers, then their customers may lose their confidence in that software system. Attackers may also target sensitive system information that can supply an attacker with the details needed to attack that system. Preventing attackers from accessing private information or obscuring that information can alleviate the risk of information leakage.

## Detailed Description Excerpts

According to Viega and McGraw [Viega 02] in Chapter 5, "Guiding Principles for Software Security," in "Principle 7: Promote Privacy" from pages 107-109:[8]

> Many users consider privacy a security concern. Try not to do anything that might compromise the privacy of the user. Be as diligent as possible in protecting any personal information a user does give your program. You've probably heard horror stories about malicious hackers accessing entire customer databases through broken web servers. It's also common to hear about attackers being able to hijack other user's shopping sessions, and thus get at private information. Try really hard to avoid the privacy doghouse. There is often no quicker way to lose customer respect than to abuse user privacy.
>
> One of the things privacy most often trades off against is usability. For example, most systems are better off forgetting about credit card numbers as soon as they are used. That way, even if the web server is compromised, there is not anything interesting stored there long-term. Users hate that solution, though, because it means they have to type in their credit card info every time they want to buy something. A more secure approach would make the convenience of "one-click shopping" impossible.
>
> The only acceptable solution in this case is to store the credit card information, but be really careful about it. We should never show the user any credit card number, even the one that belongs to the user, in case someone manages to get access they shouldn't have. A common solution is to show a partial credit-card number, with most of the digits blacked out. Though not perfect, this compromise is often acceptable.
>
> A better idea might be to ask for the issuing bank, never showing any part of the actual number once it has been captured. The next time the user wants to select a credit card, it can be done by reference to the bank. If a number needs to be changed, because there is a new card, it can be entered directly.
>
> On the server side, the credit card number should be encrypted before being stored in a database.

[3] daisy:35 (Barnum, Sean)

[4] daisy:345 (Gegick, Michael)

[8] All rights reserved. It is reprinted with permission from Addison-Wesley Professional.

Keep the key for the credit card number on a different machine (this requires decrypting and encrypting on a machine other than the one on which the database lives). That way, if the database gets compromised, then the attacker still needs to find the key, which requires breaking into another machine. This raises the bar.

User privacy isn't the only kind of privacy. Malicious hackers tend to launch attacks based on information easily collected from a hacked system. Services running on a target machine tend to give out lots of information about themselves that can help the attacker figure out how to break in. For example, the telnet service often supplies the operating system name and version:

```
$ telnet testbox
Trying 10.1.1.2...
Connected to testbox (10.1.1.2).
Escape character is '^]'.
```

```
Red Hat Linux release 6.0 (Hedwig)
Kernel 2.2.16 on an i686
login:
```

There is no reason to give out any more information than necessary. First off, a firewall should block unnecessary services, so that an attacker can get no information from them. Second, whether protected by a firewall or not (defense-in-depth, remember?), remove as much publicly provided versioning information from your software as possible. For the example above, the telnet login can be made not to advertise the operating system.

In fact, why not lie about this sort of information? It doesn't hurt to send a potential attacker on a wild goose chase by advertising a Linux box as a Solaris machine. Remote Solaris exploits don't tend to work against a Linux box. The attacker might give up in frustration long before figuring out that the service was set up to lie. (Of course, lies like this make it harder to audit your own configuration.)

Leaving any sort of information around about a system can help potential attackers. Deterring an attacker through misinformation can work. If you're using Rijndael as an encryption algorithm, does it hurt to claim to be using Twofish? Both algorithms are believed to provide similar levels of cryptographic security, so there's probably no harm done.

Attackers collecting information from a software environment can make use of all sorts of subtle information. For example, known idiosyncrasies of different web server software can quickly tell a bad guy what software a company runs, even if the software is modified not to report its version. Usually, it's impossible to close up every such information channel in a system. This type of problem is probably the most difficult security issue to identify and remove from a system. When it comes to privacy, the best strategy is to try to identify the most likely information channels, and use them to your advantage, by sending potential attackers on a wild goose chase.

Promote privacy for your users, for your systems, and for your code.

## What Goes Wrong

According to McGraw and Viega [McGraw 03a]:[11]

Finding and breaking passwords can be a trivial task.

[11] All rights reserved. It is reprinted with permission from CMP Media LLC.

Passwords are everywhere, and they're a nuisance! Say you're writing an e-commerce application that makes use of a back-end database that is protected by a DBA password. It might be tempting to hard-code the DBA credentials (username and password) into your code so you don't have to bug your user, but doing this creates a major security flaw: Now anyone who has a copy of the program has a copy of the DBA password.

Consider a security problem discovered in Netscape Communicator near the end of 1999, affecting Netscape mail users who chose to save their POP mail password with the mail client. Obviously, saving a user's mail password means storing it somewhere permanent, but where and how was the information stored? This is the sort of question that potential attackers pose all the time.

Clearly, Netscape's programmers needed to prevent casual users (including attackers) from reading the password directly off the disk, but they also had to provide access to the password for the program that used it in the POP protocol. In an attempt to solve this problem, Netscape's designers tried to encrypt the password before storing it, making it unreadable (in theory, anyway). The programmers chose a "password encryption algorithm" that they believed was good enough, considering that the source code wasn't to be made available. Unfortunately, on Windows machines, the "encrypted" password was stored in the registry, and the "crypto" was really a magic decoder-ring variant (not algorithmically strong). As a result, finding and breaking the password was a trivial task—in fact, attackers wrote a little program that decrypted actual Netscape POP passwords in a split second.

## References

[McGraw 03a] McGraw, Gary & Viega, John. "The One-Click Trick." *Software Development.* CMP Media LLC, June, 2003.

[Viega 02] Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

# Cigital, Inc. Copyright

# Velden

[1] mailto:copyright@cigital.com

| Naam | Waarde |
|---|---|
| Copyright Holder | Cigital, Inc. |

# Velden

| Naam | Waarde |
|---|---|
| is-content-area-overview | false |
| Content Areas | Knowledge/Principles |
| SDLC Relevance | Design |
| Workflow State | Publishable |